

# R workflow for forest structure metrics computation from ALS data

Jean-Matthieu Monnet, with contributions from B. Reineking and A. Glad

2021-02-01

---

Licence: CC-BY / Source page

The R code below presents a forest structure metrics computation workflow from Airborne Laser Scanning (ALS) data. Workflow is based on functions from packages **lidaRtRee** and **lidR**, packages **vegan** and **foreach** are also required. Metrics are computed for each cell of a grid defined by a resolution. Those metrics are designed to describe the 3D structure of forest. Different types of metrics are computed:

- 1D height metrics
- 2D metrics of the canopy height model (CHM)
- tree segmentation metrics (see also tree segmentation tutorial)
- forest gaps and edges metrics (see also gaps and edges detection tutorial)

The forest structure metrics derived from airborne laser scanning can be used for habitat suitability modelling and mapping. This workflow has been applied to compute the metrics used in the modeling and mapping of the habitat of Capercaillie (*Tetrao urogallus*)(Glad et al. (2020)). For more information about tree segmentation and gaps detection, please refer to the corresponding tutorials.

The workflow processes normalized point clouds provided as las/laz tiles of rectangular extent. Parallelization is used for faster processing, packages **foreach**, **future** and **doFuture** are used. A buffer is loaded around each tile to prevent border effects in tree segmentation and CHM processing.

## Parameters

Load **foreach** package and set the number of cores to use for parallel computing

```
library(foreach)
# create parallel frontend, specify to use two parallel sessions
doFuture::registerDoFuture()
future::plan("multisession", workers = 2L)
```

Numerous parameters have to be set for processing.

```
# output metrics map resolution (m)
resolution <- 10
# canopy height model resolution (m)
resCHM <- 0.5
# buffer size (m) for tile processing (20 m is better for gaps metrics, 10 m is
# enough for tree metrics)
b.size <- 20
# height threshold (m) for high points removal (points above this threshold are
# considered as outliers)
h.points <- 60
# points classes to retain for analysis (vegetation + ground)
class.points <- c(0, 1, 2, 3, 4, 5)
# ground class
class.ground <- 2
# Gaussian filter sigma values for multi-scale smoothing
```

```

sigma <- c(0, 1, 2, 4, 8, 16)
# convert vector of sigma values in list
l.sigma <- as.list(sigma)
# height breaks for penetration ratio and density
breaksH <- c(-Inf, 0, 0.5, 1, 2, 5, 10, 20, 30, 60, Inf)
# percentiles of height distribution
percent <- c(0.1, 0.25, 0.5, 0.75, 0.9)
# surface breaks for gap size (m2)
breaksGapSurface <- c(4, 16, 64, 256, 1024, 4096, Inf)
# gap surface names
n.breaksG <- gsub("-", "", paste("G.s", breaksGapSurface[c(-length(breaksGapSurface))],
  "_", breaksGapSurface[c(-1)], sep = ""))
# height bin names
n.breaksH <- gsub("-", "", paste("H.nb", breaksH[c(-length(breaksH))], "_", breaksH[c(-1)],
  sep = ""))

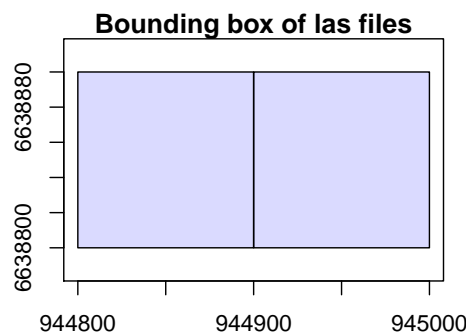
```

The first step is to create a catalog of LAS files (should be normalized, non-overlapping rectangular tiles). Preferably, tiles should be aligned on a multiple of resolution, and points should not lie on the northern or eastern border when such borders are common with adjacent tiles.

```

# create catalog of LAS files
cata <- lidR::catalog("./data/forest.structure.metrics")
# set coordinate system
sp::proj4string(cata) <- sp::CRS(SRS_string = "EPSG:2154")
# set sensor type
lidR::sensor(cata) <- "ALS"
# disable display of catalog processing
lidR::opt_progress(cata) <- FALSE
# display
sp::plot(cata, main = "Bounding box of las files")

```



## Workflow exemplified on one tile

### Load point cloud

The tile is loaded, plus a buffer on adjacent tiles. Buffer points have a column “buffer” filled with 1.

```

# tile to process
i <- 1
# tile extent
b.box <- as.numeric(cata@data[i, c("Min.X", "Min.Y", "Max.X", "Max.Y")])
# load tile extent plus buffer
a <- lidR::clip_rectangle(cata, b.box[1] - b.size, b.box[2] - b.size, b.box[3] +
  b.size, b.box[4] + b.size)
# add 'buffer' flag to points in buffer with TRUE value in this new attribute
a <- lidR::add_attribute(a, a$X < b.box[1] | a$Y < b.box[2] | a$X >= b.box[3] | a$Y >=
  b.box[4], "buffer")
a

```

```
## class      : LAS (v1.1 format 1)
## memory     : 26.7 Mb
## extent     : 944800, 944920, 6638800, 6638900 (xmin, xmax, ymin, ymax)
## coord. ref. : RGF93 / Lambert-93
## area       : 11996.4 m2
## points     : 333.3 thousand points
## density    : 27.78 points/m2
```

Only points from desired classes are retained. In case some mis-classified, high or low points remain in the data set, the point cloud is filtered and negative heights are replaced by 0.

```
# remove unwanted point classes, and points higher than height threshold
a <- lidR::filter_poi(a, is.element(Classification, class.points) & Z <= h.points)
# set negative heights to 0
a@data$Z[a@data$Z < 0] <- 0
#
summary(a@data)
```

```
##           X                Y                Z                gpstime
## Min.      :944800    Min.      :6638800    Min.      : 0.00    Min.      :1.52e+08
## 1st Qu.:944828    1st Qu.:6638825    1st Qu.: 2.33    1st Qu.:1.52e+08
## Median :944860    Median :6638850    Median : 9.94    Median :1.52e+08
## Mean    :944860    Mean    :6638849    Mean    :10.90    Mean    :1.52e+08
## 3rd Qu.:944891    3rd Qu.:6638872    3rd Qu.:18.14    3rd Qu.:1.52e+08
## Max.    :944920    Max.    :6638900    Max.    :37.45    Max.    :1.52e+08
## Intensity      ReturnNumber    NumberOfReturns    ScanDirectionFlag
## Min.      : 2.0    Min.      :1.000    Min.      :1.000    Min.      :0
## 1st Qu.: 27.0    1st Qu.:1.000    1st Qu.:2.000    1st Qu.:0
## Median : 67.0    Median :1.000    Median :2.000    Median :0
## Mean    :101.7    Mean    :1.713    Mean    :2.435    Mean    :0
## 3rd Qu.:135.0    3rd Qu.:2.000    3rd Qu.:3.000    3rd Qu.:0
## Max.    :821.0    Max.    :7.000    Max.    :7.000    Max.    :0
## EdgeOfFlightline Classification    Synthetic_flag    Keypoint_flag
## Min.      :0      Min.      :0.000    Mode :logical    Mode :logical
## 1st Qu.:0      1st Qu.:5.000    FALSE:333229    FALSE:333229
## Median :0      Median :5.000
## Mean    :0      Mean    :4.122
## 3rd Qu.:0      3rd Qu.:5.000
## Max.    :0      Max.    :5.000
## Withheld_flag    ScanAngleRank    UserData    PointSourceID    buffer
## Mode :logical    Min.      :-2.00    Min.      :0      Min.      :0      Mode :logical
## FALSE:333229    1st Qu.: 1.00    1st Qu.:0      1st Qu.:0      FALSE:278011
## Median :86.00    Median :0      Median :0      TRUE :55218
## Mean    :44.68    Mean    :0      Mean    :0
## 3rd Qu.:87.00    3rd Qu.:0      3rd Qu.:0
## Max.    :87.00    Max.    :0      Max.    :0
```

## Canopy height model

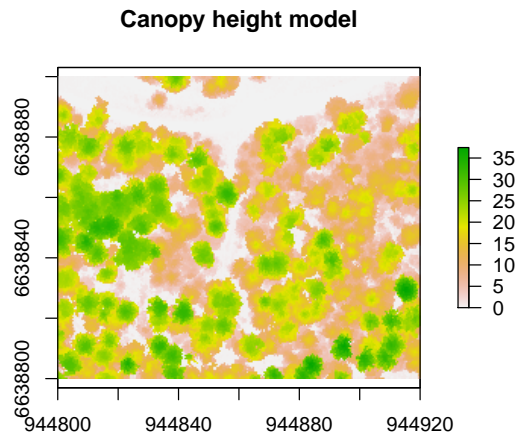
The next step is to compute the canopy height model (CHM). It will be used to derive:

- 2D canopy height metrics related to multi-scale vertical heterogeneity (mean and standard deviation of CHM, smoothed at different scales)
- tree metrics from tree top segmentation
- gaps and edges metrics from gap segmentation

The CHM is computed and NA values are replaced by 0. A check is performed to make sure low or high points are not present.

```
# compute chm
chm <- lidaRtRee::points2DSM(a, res = resCHM)
```

```
# replace NA, low and high values
chm[is.na(chm) | chm < 0 | chm > h.points] <- 0
# display CHM
raster::plot(chm, asp = 1, main = "Canopy height model")
```

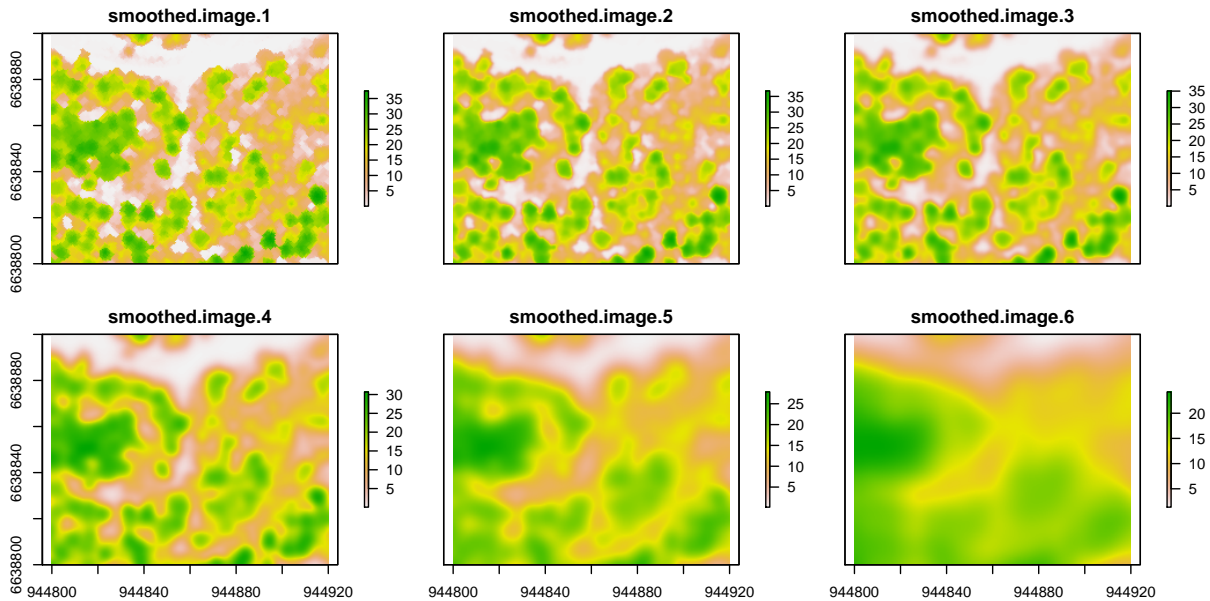


## Metrics computation

### 2D CHM metrics

The CHM is smoothed with a Gaussian filter with different **sigma** values. Smoothed results are stored in a list and then converted to a raster stack.

```
# for each value in list of sigma, apply filtering to chm and store result in
# list
st <- lapply(l.sigma, FUN = function(x) {
  lidaRtRee::demFiltering(chm, nlFilter = "Closing", nlSize = 5, sigmap = x)[[2]]
})
# convert to raster
st <- raster::stack(st)
# display
raster::plot(st)
```



The raster stack is then converted to points. Points outside the tile extent are removed and summary statistics (mean and standard deviation) of smoothed CHM heights are computed for each pixel at the output metrics resolution. Canopy cover in different height layers are also computed for the initial CHM after applying a non-linear filter designed to fill holes.

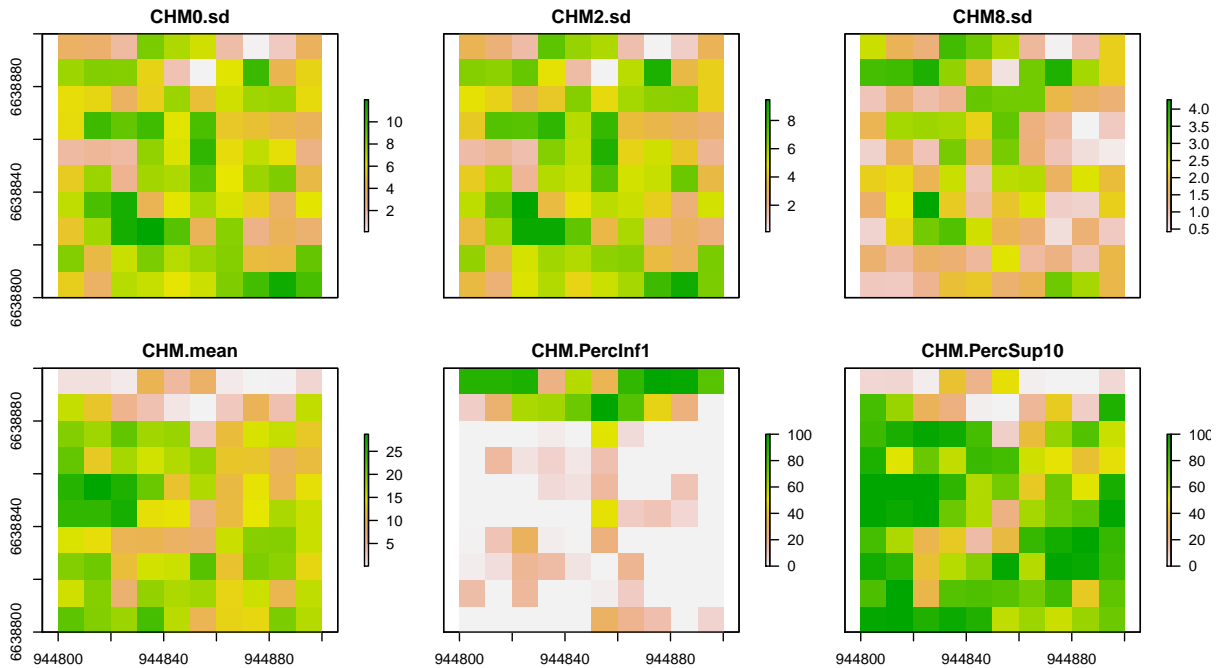
```
# crop to bbox
st <- raster::crop(st, raster::extent(b.box[1], b.box[3], b.box[2], b.box[4]))
# multiplying factor to compute proportion
mf <- 100/(resolution/resCHM)^2
# modify layer names
names(st) <- paste0("layer.", 1:6)
# compute raster metrics
metrics.2dchm <- lidaRtRee::rasterMetrics(st, res = resolution, fun = function(x) {
  c(sd(x$layer.1), sd(x$layer.2), sd(x$layer.3), sd(x$layer.4), sd(x$layer.5),
    sd(x$layer.6), mean(x$layer.1), sum(x$layer.1 < 0.5) * mf, sum(x$layer.1 <
      1) * mf, sum(x$layer.1 > 5) * mf, sum(x$layer.1 > 10) * mf, sum(x$layer.1 >
      20) * mf, (sum(x$layer.1 < 5) - sum(x$layer.1 < 1)) * mf, (sum(x$layer.1 <
      5) - sum(x$layer.1 < 2)) * mf)
}, output = "raster")
names(metrics.2dchm) <- c("CHM0.sd", "CHM1.sd", "CHM2.sd", "CHM4.sd", "CHM8.sd",
  "CHM16.sd", "CHM.mean", "CHM.PercInf0.5", "CHM.PercInf1", "CHM.PercSup5", "CHM.PercSup10",
  "CHM.PercSup20", "CHM.Perc1_5", "CHM.Perc2_5")
#
metrics.2dchm

## class      : RasterBrick
## dimensions  : 10, 10, 100, 14  (nrow, ncol, ncell, nlayers)
## resolution  : 10, 10  (x, y)
## extent     : 944800, 944900, 6638800, 6638900  (xmin, xmax, ymin, ymax)
## crs        : +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000 +y_0=6600000 +ellps=G
## source     : memory
## names      : CHM0.sd, CHM1.sd, CHM2.sd, CHM4.sd, CHM8.sd, CHM16.sd, CHM.mean,
## min values : 0.09293137, 0.08238094, 0.12682748, 0.21107244, 0.42359988, 0.17968139, 0.08870000,
## max values : 11.990108, 10.747519, 9.463671, 7.492695, 4.271958, 2.322395, 28.713500,
```

Some outputs are displayed hereafter. On the first line are the standard deviation of CHM smoothed with different sigma values. On the second line are the CHM mean and percentages of CHM values below 1 m and above 10 m.

```
raster::plot(metrics.2dchm[[c("CHM0.sd", "CHM2.sd", "CHM8.sd", "CHM.mean", "CHM.PercInf1",
```

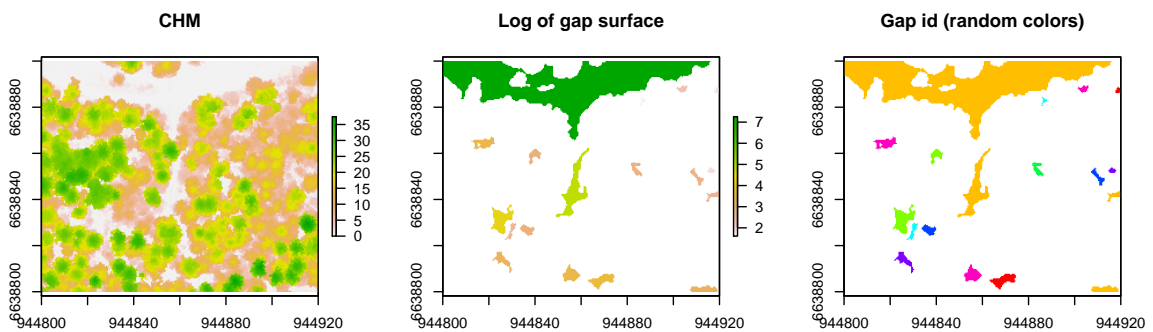
```
"CHM.PercSup10")]])
```



## Gaps and edges metrics

Gaps are computed with the function `gapDetection`.

```
# compute gaps
gaps <- lidaRtRee::gapDetection(chm, ratio = 2, gap.max.height = 1, min.gap.surface = min(breaksGapS
  closing.height.bin = 1, nlFilter = "Median", nlsize = 3, gapReconstruct = TRUE)
# display maps
par(mfrow = c(1, 3))
raster::plot(chm, main = "CHM")
raster::plot(log(gaps$gap.surface), main = "Log of gap surface")
# display gaps
dummy <- gaps$gap.id
dummy[dummy == 0] <- NA
raster::plot(dummy%%8, asp = 1, main = "Gap id (random colors)", col = rainbow(8),
  legend = FALSE)
```



Summary statistics are then computed: pixel surface occupied by gaps in different size classes. Results are first cropped to the tile extent.

```

# crop results to tile size
gaps.surface <- raster::crop(gaps$gap.surface, raster::extent(b.box[1], b.box[3],
  b.box[2], b.box[4]))
# compute gap statistics at final metrics resolution, in case gaps exist
if (!all(is.na(raster::values(gaps.surface)))) {
  metrics.gaps <- lidaRtRee::rasterMetrics(gaps.surface, res = resolution, fun = function(x) {
    hist(x$layer, breaks = breaksGapSurface, plot = F)$counts * (resCHM/resolution)^2
  }, output = "raster")
  # compute total gap proportion
  metrics.gaps$sum <- raster::stackApply(metrics.gaps, rep(1, length(names(metrics.gaps))),
    fun = sum)
  # if gaps are present
  names(metrics.gaps) <- c(n.breaksG, paste("G.s", min(breaksGapSurface), "_Inf",
    sep = ""))
  # replace possible NA values by 0 (NA values are present in lines of the target
  # raster where no values >0 are present in the input raster)
  metrics.gaps[is.na(metrics.gaps)] <- 0
} else {
  metrics.gaps <- NULL
}
#
metrics.gaps

## class      : RasterBrick
## dimensions  : 10, 10, 100, 7  (nrow, ncol, ncell, nlayers)
## resolution  : 10, 10  (x, y)
## extent     : 944800, 944900, 6638800, 6638900  (xmin, xmax, ymin, ymax)
## crs        : +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000 +y_0=6600000 +ellps=G
## source      : memory
## names       : G.s4_16, G.s16_64, G.s64_256, G.s256_1024, G.s1024_4096, G.s4096_Inf, G.s4_Inf
## min values  :      0,      0,      0,      0,      0,      0,      0
## max values  : 0.0750, 0.3200, 0.5175, 0.0000, 1.0000, 0.0000, 1.0000

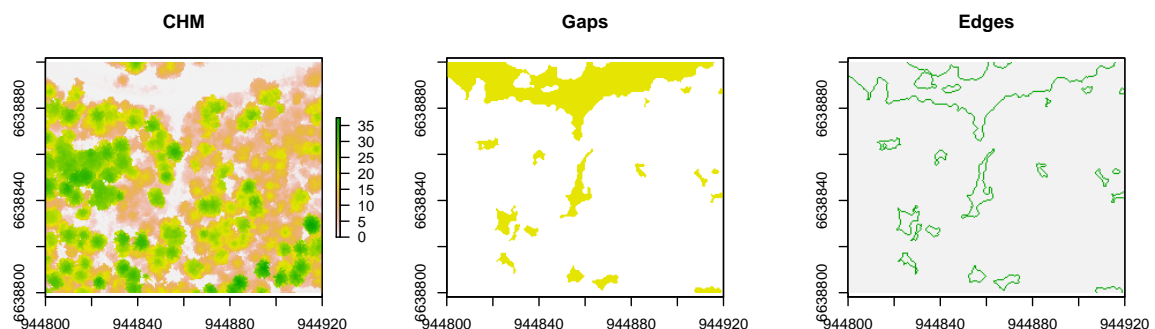
```

Edges are detected with the function `edgeDetection` as the outer envelope of the previously delineated gaps.

```

# Perform edge detection by erosion
edges <- lidaRtRee::edgeDetection(gaps$gap.id > 0)
# display maps
par(mfrow = c(1, 3))
raster::plot(chm, main = "CHM")
raster::plot(gaps$gap.id > 0, main = "Gaps", legend = FALSE)
raster::plot(edges, main = "Edges", legend = FALSE)

```



The percentage of surface occupied by edges is then computed as summary statistic. Results are first



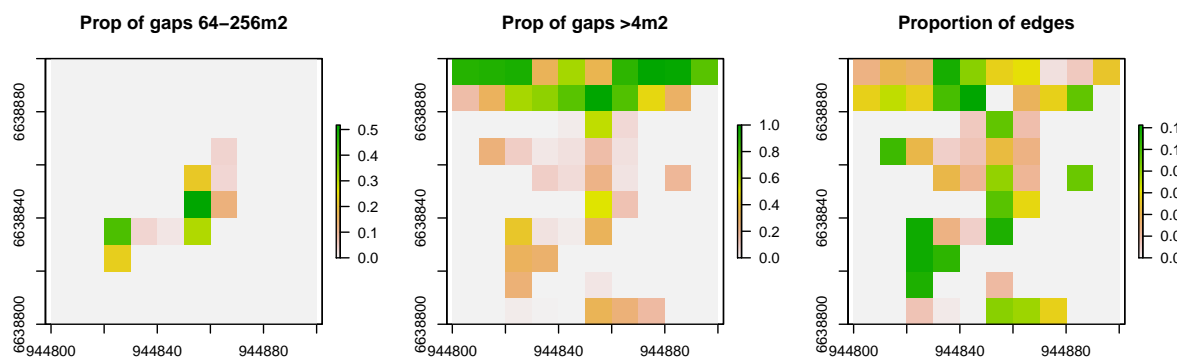
cropped to the tile extent.

```
# crop results to tile size
edges <- raster::crop(edges, raster::extent(b.box[1], b.box[3], b.box[2], b.box[4]))
# compute gap statistics at final metrics resolution, in case gaps exist
if (!all(is.na(raster::values(edges)))) {
  metrics.edges <- lidaRtRee::rasterMetrics(edges, res = resolution, fun = function(x) {
    sum(x$layer) * (resCHM/resolution)^2
  }, output = "raster")
  names(metrics.edges) <- "edges.proportion"
  # replace possible NA values by 0 (NA values are present in lines of the target
  # raster where no values >0 are present in the input raster)
  metrics.edges[is.na(metrics.edges)] <- 0
} else {
  metrics.edges <- NULL
}
#
metrics.edges

## class      : RasterLayer
## dimensions : 10, 10, 100 (nrow, ncol, ncell)
## resolution : 10, 10 (x, y)
## extent     : 944800, 944900, 6638800, 6638900 (xmin, xmax, ymin, ymax)
## crs        : +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000 +y_0=6600000 +ellps=G
## source     : memory
## names      : edges.proportion
## values     : 0, 0.1225 (min, max)
```

Some outputs are displayed hereafter. The proportion of surface occupied by gaps of various sizes is depicted as well as the proportion of surface occupied by edges.

```
par(mfrow = c(1, 3))
# raster::plot(metrics.gaps[['G.s4_16']], main='Prop of gaps 4-16m2')
raster::plot(metrics.gaps[['G.s64_256']], main = "Prop of gaps 64-256m2")
raster::plot(metrics.gaps[['G.s4_Inf']], main = "Prop of gaps >4m2")
raster::plot(metrics.edges, main = "Proportion of edges")
```



## Tree metrics

Tree tops are detected with the function `treeSegmentation` and then extracted with `treeExtraction`.

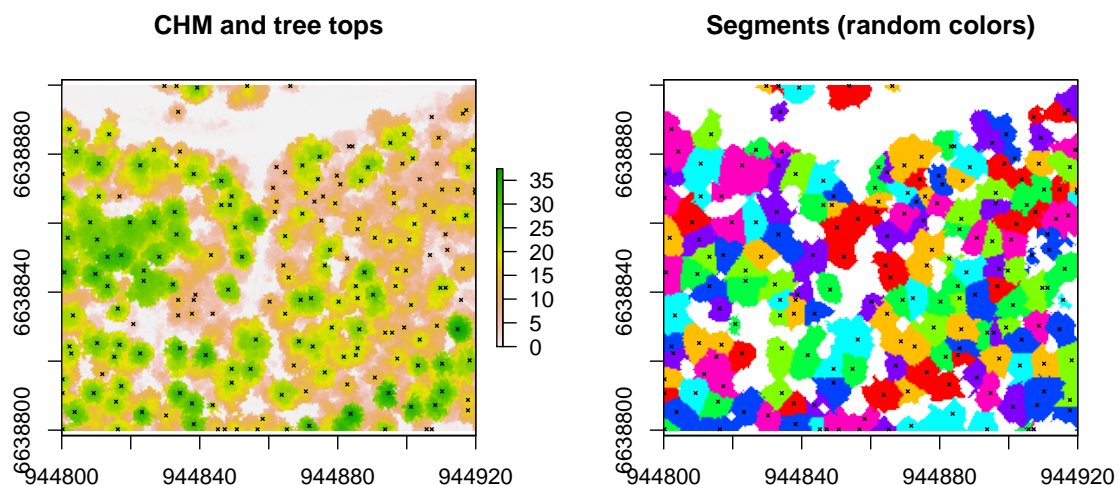
```
# tree top detection (default parameters)
segms <- lidaRtRee::treeSegmentation(chm, hmin = 5)
# extraction to data.frame
trees <- lidaRtRee::treeExtraction(segms$filled.dem, segms$local.maxima, segms$segments.id)
```



```

# display maps
par(mfrow = c(1, 2))
raster::plot(chm, main = "CHM and tree tops")
points(trees$x, trees$y, pch = 4, cex = 0.3)
# display segments, except ground segment
dummy <- segms$segments.id
dummy[dummy == 0] <- NA
raster::plot(dummy%%8, asp = 1, main = "Segments (random colors)", col = rainbow(8),
  legend = FALSE)
points(trees$x, trees$y, pch = 4, cex = 0.3)

```



Results are cropped to the tile extent. Summary statistics from `stdTreeMetrics` are then computed for each pixel. Canopy cover in trees and mean canopy height in trees are computed in a second step because they can not be computed from the data of the extracted trees which crowns may span several output pixels. They are calculated from the images obtained during the previous tree segmentation.

```

# remove trees outside of tile
trees <- trees[trees$x >= b.box[1] & trees$x < b.box[3] & trees$y >= b.box[2] & trees$y <
  b.box[4], ]
# compute raster metrics
metrics.trees <- lidaRtRee::rasterMetrics(trees[, -1], res = resolution, fun = function(x) {
  lidaRtRee::stdTreeMetrics(x, resolution^2/10000)
}, output = "raster")
# remove NAs and NaNs
metrics.trees[!is.finite(metrics.trees)] <- 0
# remove missing variables
metrics.trees <- raster::dropLayer(metrics.trees, c("Tree.CanopyCover", "Tree.meanCanopyH"))
# compute canopy cover in trees and canopy mean height in trees, because it is
# not correctly calculated in previous step.
r.treechm <- segms$filled.dem
# set chm to 0 in non segment area
r.treechm[segms$segments.id == 0] <- NA
# compute raster metrics
dummy <- lidaRtRee::rasterMetrics(raster::crop(r.treechm, raster::extent(b.box[1],
  b.box[3], b.box[2], b.box[4])), res = resolution, fun = function(x) {
  c(sum(!is.na(x$filled.dem)), sum(x$filled.dem, na.rm = T))/(resolution/resCHM)^2
}, output = "raster")
names(dummy) <- c("Tree.CanopyCover", "Tree.meanCanopyH")

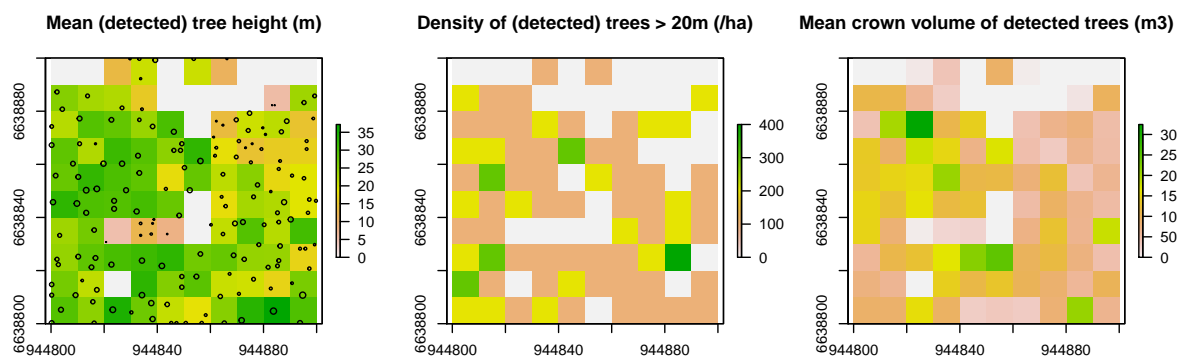
```

```
#
dummy <- raster::extend(dummy, metrics.trees)
metrics.trees <- raster::addLayer(metrics.trees, dummy)
#
metrics.trees

## class      : RasterStack
## dimensions : 10, 10, 100, 13  (nrow, ncol, ncell, nlayers)
## resolution : 10, 10  (x, y)
## extent      : 944800, 944900, 6638800, 6638900  (xmin, xmax, ymin, ymax)
## crs         : +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000 +y_0=6600000 +ellps=G
## names       : Tree.meanH, Tree.sdH, Tree.giniH, Tree.density, TreeInf10.density, TreeSup10
## min values  : 0, 0, 0, 0, 0, 0
## max values  : 37.1800000, 15.1957247, 0.2050181, 500.0000000, 200.0000000, 500
```

Some outputs are displayed hereafter.

```
par(mfrow = c(1, 3))
raster::plot(metrics.trees$Tree.meanH, main = "Mean (detected) tree height (m)")
points(trees$x, trees$y, cex = trees$h/40)
raster::plot(metrics.trees$TreeSup20.density, main = "Density of (detected) trees > 20m (/ha)")
raster::plot(metrics.trees$Tree.meanCrownVolume, main = "Mean crown volume of detected trees (m3)")
```



## Point cloud (1D) metrics

Buffer points are first removed from the point cloud, then metrics are computed from all points and from vegetation points only.

```
# remove buffer points
a <- lidR::filter_poi(a, buffer == 0)
# all points metrics
metrics.1d <- lidR::grid_metrics(a, as.list(c(as.vector(quantile(Z, probs = percent)),
  sum(Classification == 2), mean(Intensity[ReturnNumber == 1]))), res = resolution)
names(metrics.1d) <- c(paste("H.p", percent * 100, sep = ""), "nbSol", "I.mean.1stpulse")
# vegetation-only metrics
a <- lidR::filter_poi(a, Classification != class.ground)
dummy <- lidR::grid_metrics(a, as.list(c(mean(Z), max(Z), sd(Z), length(Z), hist(Z,
  breaks = breaksH, right = F, plot = F)$counts)), res = resolution)
names(dummy) <- c("H.mean", "H.max", "H.sd", "nbVeg", n.breaksH)
# merge rasterstacks
dummy <- raster::extend(dummy, metrics.1d)
metrics.1d <- raster::addLayer(metrics.1d, dummy)
rm(dummy)
# crop to tile extent
metrics.1d <- raster::crop(metrics.1d, raster::extent(b.box[1], b.box[3], b.box[2],
```

```
b.box[4]))
```

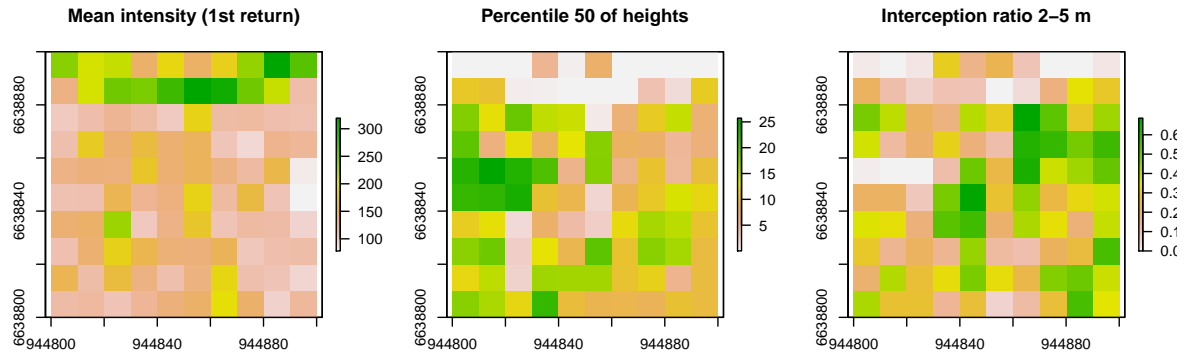
Based on those metrics, additional metrics are computed (Simpson index, relative density in height bins and penetration ratio in height bins)

```
# Simpson index
metrics.1d$H.simpson <- raster::stackApply(metrics.1d[[n.breaksH[c(-1, -length(n.breaksH))]]],
  1, function(x, ...) {
    vegan::diversity(x, index = "simpson")
  })
# Relative density
for (i in n.breaksH[c(-1, -length(n.breaksH))]) {
  metrics.1d[[paste0(i, "relative_density")]] <- metrics.1d[[i]]/(metrics.1d$nbVeg +
    metrics.1d$nbSol)
}
# Penetration ratio cumulative sum
cumu.sum <- metrics.1d[["nbSol"]]
for (i in n.breaksH) {
  cumu.sum <- raster::addLayer(cumu.sum, cumu.sum[[dim(cumu.sum)[3]]] + metrics.1d[[i]])
}
names(cumu.sum) <- c("nbSol", n.breaksH)
# compute interception ratio of each layer
intercep.ratio <- cumu.sum[[-1]]
for (i in 2:dim(cumu.sum)[3]) {
  intercep.ratio[[i - 1]] <- 1 - cumu.sum[[i - 1]]/cumu.sum[[i]]
}
names(intercep.ratio) <- paste0(names(cumu.sum)[-1], "ratio")
# merge rasterstacks
metrics.1d <- raster::addLayer(metrics.1d, intercep.ratio)
# rm(cumu.sum, intercep.ratio)
metrics.1d
```

```
## class      : RasterStack
## dimensions : 10, 10, 100, 40  (nrow, ncol, ncell, nlayers)
## resolution : 10, 10  (x, y)
## extent     : 944800, 944900, 6638800, 6638900  (xmin, xmax, ymin, ymax)
## crs        : +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000 +y_0=6600000 +ellps=G
## names      : H.p10, H.p25, H.p50, H.p75, H.p90, nbSol,
## min values : 0.000000e+00, 0.000000e+00, 1.000000e-02, 3.000000e-02, 6.000000e-02, 5.200000e+01,
## max values : 1.762000e+01, 2.269000e+01, 2.572500e+01, 2.791000e+01, 3.167000e+01, 9.820000e+02,
```

Some outputs are displayed hereafter.

```
# display results
par(mfrow = c(1, 3))
raster::plot(metrics.1d$I.mean.1stpulse, main = "Mean intensity (1st return)")
raster::plot(metrics.1d$H.p50, main = "Percentile 50 of heights")
raster::plot(metrics.1d$H.nb2_5ratio, main = "Interception ratio 2-5 m")
```



## Batch processing

The following code uses parallel processing to handle multiple files and arranges all metrics in a raster stack. Code in the “parameters” section has to be run beforehand.

```
# processing by tile
metrics <- foreach::foreach (i=1:nrow(cata), .errorhandling="remove") %dopar%
{
  # tile extent
  b.box <- as.numeric(cata@data[i, c("Min.X", "Min.Y", "Max.X", "Max.Y")])
  # load tile extent plus buffer
  a <- try(lidR::clip_rectangle(cata, b.box[1]-b.size, b.box[2]-b.size, b.box[3]+b.size,
                              b.box[4]+b.size))

  #
  if (class(a)=="try-error") # check if points are successfully loaded
  {
    # else NULL output
    temp <- NULL
  } else {
    # add 'buffer' flag to points in buffer with TRUE value in this new attribute
    a <- lidR::add_attribute(a,
                           a$X < b.box[1] | a$Y < b.box[2] | a$X >= b.box[3] | a$Y >= b.box[4],
                           "buffer")

    # remove unwanted point classes, and points higher than height threshold
    a <- lidR::filter_poi(a, is.element(Classification, class.points) & Z<=h.points)
    # check number of remaining points
    if (length(a)==0)
    {
      # output NULL
      temp <- NULL
    } else {
      # set negative heights to 0
      a@data$Z[a@data$Z<0] <- 0
      #
      # compute chm
      chm <- lidaRtRee::points2DSM(a, res=resCHM)
      # replace NA, low and high values
      chm[is.na(chm) | chm<0 | chm>h.points] <- 0
      # #####
      # compute 2D CHM metrics
      # for each value in list of sigma, apply filtering to chm and store result in list
      st <- lapply(l.sigma, FUN = function(x)
      {
        lidaRtRee::demFiltering(chm, nlFilter = "Closing", nlSize = 5, sigmap = x)[[2]]
      })
    }
  }
}
```

```

}))
# convert to raster
st <- raster::stack(st)
# crop to bbox
st <- raster::crop(st, raster::extent(b.box[1], b.box[3], b.box[2], b.box[4]))
# multiplying factor to compute proportion
mf <- 100/(resolution/resCHM)^2
# compute raster metrics
metrics.2dchm <-
  lidaRtRee::rasterMetrics(
    st,
    res = resolution,
    fun = function(x) {
      data.frame(
        CHM0.sd = sd(x$non.linear.image),
        CHM1.sd = sd(x$smoothed.image.1),
        CHM2.sd = sd(x$smoothed.image.2),
        CHM4.sd = sd(x$smoothed.image.4),
        CHM8.sd = sd(x$smoothed.image.8),
        CHM16.sd = sd(x$smoothed.image.16),
        CHM.mean = mean(x$non.linear.image),
        CHM.PercInf0.5 = sum(x$non.linear.image < 0.5) * mf,
        CHM.PercInf1 = sum(x$non.linear.image < 1) * mf,
        CHM.PercSup5 = sum(x$non.linear.image > 5) * mf,
        CHM.PercSup10 = sum(x$non.linear.image > 10) * mf,
        CHM.PercSup20 = sum(x$non.linear.image > 20) * mf,
        CHM.Perc1_5 = (sum(x$non.linear.image < 5) - sum(x$non.linear.image < 1)) * mf,
        CHM.Perc2_5 = (sum(x$non.linear.image < 5) - sum(x$non.linear.image < 2)) * mf
      )
    },
    output="raster")
#
# #####
# compute gap metrics
gaps <- lidaRtRee::gapDetection(chm, ratio=2, gap.max.height=1,
                               min.gap.surface=min(breaksGapSurface),
                               closing.height.bin=1,
                               nlFilter="Median", nlsize=3,
                               gapReconstruct=TRUE)
# crop results to tile size
gaps.surface <- raster::crop(gaps$gap.surface, raster::extent(b.box[1], b.box[3], b.box[2],
                                                             b.box[4]))
# compute gap statistics at final metrics resolution, in case gaps exist
if (!all(is.na(raster::values(gaps.surface))))
{
  metrics.gaps <- lidaRtRee::rasterMetrics(gaps.surface, res=resolution,
                                           fun=function(x){
                                             hist(x$layer, breaks=breaksGapSurface,
                                                  plot=F)$counts*(resCHM/resolution)^2
                                           },
                                           output="raster")
# compute total gap proportion
metrics.gaps$sum <- raster::stackApply(metrics.gaps, rep(1,length(names(metrics.gaps))),
                                       fun = sum)
# if gaps are present
names(metrics.gaps) <- c(n.breaksG, paste("G.s", min(breaksGapSurface), "_Inf", sep=""))
# replace possible NA values by 0 (NA values are present in lines of the target raster

```

```

    # where no values >0 are present in the input raster)
    metrics.gaps[is.na(metrics.gaps)] <- 0
} else {metrics.gaps <- NULL}
#
#####
# compute edge metrics
# Perform edge detection by erosion
edges <- lidaRtRee::edgeDetection(gaps$gap.id>0)
# crop results to tile size
edges <- raster::crop(edges, raster::extent(b.box[1], b.box[3],
                                             b.box[2], b.box[4]))
# compute edges statistics at final metrics resolution, in case edges exist
if (!all(is.na(raster::values(edges))))
{
    metrics.edges <- lidaRtRee::rasterMetrics(edges,
                                              res=resolution,
                                              fun=function(x)
                                              {
                                                  sum(x$layer)*(resCHM/resolution)^2
                                              }, output="raster")

    names(metrics.edges) <- "edges.proportion"
    # replace possible NA values by 0 (NA values are present in lines of the target raster
    # where no values >0 are present in the input raster)
    metrics.edges[is.na(metrics.edges)] <- 0
} else {metrics.edges <- NULL}
#
#####
# compute tree metrics
# tree top detection (default parameters)
segms <- lidaRtRee::treeSegmentation(chm, hmin=5)
# extraction to data.frame
trees <- lidaRtRee::treeExtraction(segms$filled.dem, segms$local.maxima, segms$segments.id)
# remove trees outside of tile
trees <- trees[trees$x>=b.box[1] & trees$x<b.box[3] & trees$y>=b.box[2] & trees$y<b.box[4],]
# compute raster metrics
metrics.trees <- lidaRtRee::rasterMetrics(trees[, -1], res=resolution,
                                          fun=function(x){
                                              lidaRtRee::stdTreeMetrics(x, resolution^2/10000)
                                          },
                                          output="raster")

# remove NAs and NaNs
metrics.trees[!is.finite(metrics.trees)] <- 0
# remove missing variables
metrics.trees <- raster::dropLayer(metrics.trees, c("Tree.CanopyCover", "Tree.meanCanopyH"))
#
# extend layer in case there are areas without trees
metrics.trees <- raster::extend(metrics.trees, metrics.2dchm, values=0)
# compute canopy cover in trees and canopy mean height in trees, because it is not correctly
# calculated in previous step.
r.treechm <- segms$filled.dem
# set chm to 0 in non segment area
r.treechm[segms$segments.id==0] <- NA
# compute raster metrics
dummy <- lidaRtRee::rasterMetrics(
    raster::crop(r.treechm, raster::extent(b.box[1], b.box[3], b.box[2], b.box[4])),
    res=resolution,
    fun=function(x){

```

```

        c(sum(!is.na(x$filled.dem)),
          sum(x$filled.dem,na.rm=T))/(resolution/resCHM)^2
      }
      , output="raster")
names(dummy) <- c("Tree.CanopyCover", "Tree.meanCanopyH")
dummy <- raster::extend(dummy, metrics.2dchm, values=0)
#
metrics.trees <- raster::addLayer(metrics.trees, dummy)
#
#####
# compute 1D height metrics
# remove buffer points
a <- lidR::filter_poi(a, buffer==0)
#
if (length(a)==0)
{
  metrics.1d <- NULL
} else {
  # all points metrics
  metrics.1d <- lidR::grid_metrics(a, as.list(c(as.vector(quantile(Z,probs=percent)),
                                              sum(Classification==2),
                                              mean(Intensity [ReturnNumber == 1])
))),res = resolution)
names(metrics.1d) <- c(paste("H.p",percent*100,sep=""),"nbSol","I.mean.1stpulse")
#
# vegetation-only metrics
a <- lidR::filter_poi(a, Classification!=class.ground)
if (length(a)!=0)
{
  dummy <- lidR::grid_metrics(a, as.list(c(mean(Z),
                                          max(Z),
                                          sd(Z),
                                          length(Z),
                                          hist(Z, breaks=breaksH, right=F, plot=F)$counts)
))),res = resolution)
names(dummy) <- c("H.mean","H.max","H.sd","nbVeg",n.breaksH)
# merge rasterstacks
dummy <- raster::extend(dummy, metrics.1d)
metrics.1d <- raster::addLayer(metrics.1d, dummy)
rm(dummy)
#####
# merge metrics
metrics.1d <- raster::extend(metrics.1d, metrics.2dchm, values=0)
metrics.1d <- raster::crop(metrics.1d, metrics.2dchm)
metrics.gaps <- raster::extend(metrics.gaps, metrics.2dchm, values=0)
metrics.gaps <- raster::crop(metrics.gaps, metrics.2dchm)
metrics.edges <- raster::extend(metrics.edges, metrics.2dchm, values=0)
metrics.edges <- raster::crop(metrics.edges, metrics.2dchm)
metrics.trees <- raster::crop(metrics.trees, metrics.2dchm)
#
temp <- raster::addLayer(metrics.1d, metrics.2dchm, metrics.gaps,
                        metrics.edges, metrics.trees)
temp[is.na(temp)] <- 0
} # end of vegetation-only points presence check
} # end of buffer-less points presence check
} # end of buffered area points presence check
} # success of file reading check

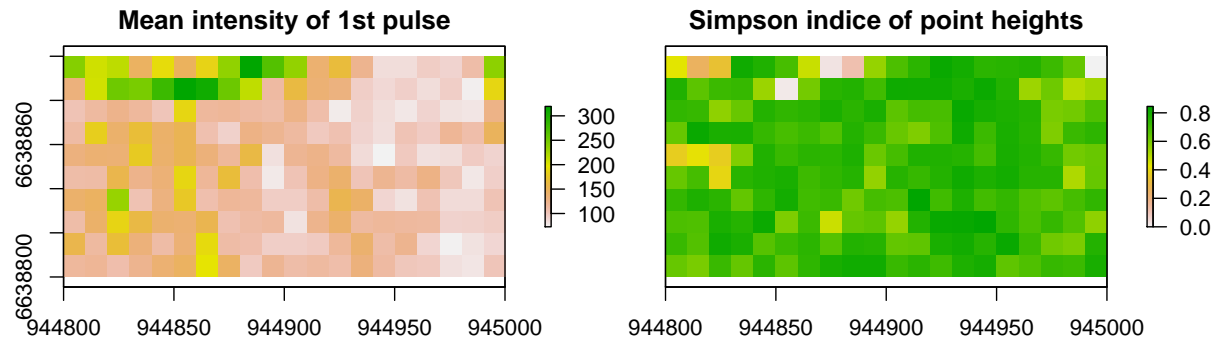
```



```

    return(temp)
}
# #####
# mosaic rasters
names.metrics <- names(metrics[[1]])
metrics <- do.call(raster::merge, metrics)
names(metrics) <- names.metrics
# #####
# compute additional metrics
# Simpson index
metrics$H.simpson <- raster::stackApply(metrics[[n.breaksH[c(-1,-length(n.breaksH))]]], 1,
                                         function(x, ...){vegan::diversity(x, index="simpson")})
#
# Relative density
for (i in n.breaksH[c(-1,-length(n.breaksH))])
{
  metrics[[paste0(i,"relative_density")]] <- metrics[[i]] / (metrics$nbVeg + metrics$nbSol)
}
# Penetration ratio
# compute cumulative sum
cum.sum <- metrics[["nbSol"]]
for (i in n.breaksH)
{
  cum.sum <- raster::addLayer(cum.sum, cum.sum[[dim(cum.sum)[3]]] + metrics[[i]])
}
names(cum.sum) <- c("nbSol", n.breaksH)
# interception ratio
intercep.ratio <- cum.sum[[-1]]
for (i in 2:dim(cum.sum)[3])
{
  intercep.ratio[[i-1]] <- 1 - cum.sum[[i-1]]/cum.sum[[i]]
}
names(intercep.ratio) <- paste0(names(cum.sum)[-1], "ratio")
# merge rasterstacks
metrics <- raster::addLayer(metrics, intercep.ratio)
#
# #####
# export as raster files
# for (i in 1:names(metrics))
# {
#   print(i)
#   writeRaster(metrics[[i]],file=paste("output/raster_",i,"_",resolution,"m.tif",sep="")
# }
# #####
# display
raster::plot(metrics[[c("I.mean.1stpulse","H.simpson")]],
              main=c("Mean intensity of 1st pulse","Simpson indice of point heights"))

```



## References

- Glad, Anouk, Björn Reineking, Marc Montadert, Alexandra Depraz, and Jean-Matthieu Monnet. 2020. "Assessing the Performance of Object-Oriented LiDAR Predictors for Forest Bird Habitat Suitability Modeling." *Remote Sensing in Ecology and Conservation* 6 (1): 5–19. <https://doi.org/10.1002/rse2.117>.